
MODULE *DistributedLock*

EXTENDS *Naturals, FiniteSets, Sequences, TLC*

The set of clients

CONSTANT *Clients*

Client states

CONSTANTS *Active, Inactive*

Message types

CONSTANT *LockRequest, LockResponse, TryLockRequest, TryLockResponse, UnlockRequest, UnlockResponse*

An empty constant

CONSTANT *Nil*

The current lock holder

VARIABLE *lock*

The lock queue

VARIABLE *queue*

The current lock *ID*

VARIABLE *id*

$serverVars \triangleq \langle lock, id, queue \rangle$

Client states

VARIABLE *clients*

$clientVars \triangleq \langle clients \rangle$

Client messages

VARIABLE *messages*

Variable

VARIABLE *messageCount*

$messageVars \triangleq \langle messages, messageCount \rangle$

The invariant checks that:

- * No client can hold more than one lock at a time
- * No two clients hold a lock with the same *ID*
- * The lock is held by an active session

Note that more than one client may believe itself to hold the lock at the same time, *e.g.* if a client's session has expired but the client hasn't been notified, but lock *IDs* must be unique and monotonically increasing.

$TypeInvariant \triangleq$

$$\begin{aligned}
& \wedge \forall c \in \text{DOMAIN } clients : \text{Cardinality}(clients[c].locks) \in 0 .. 1 \\
& \wedge \forall c1, c2 \in \text{DOMAIN } clients : c1 \neq c2 \Rightarrow \text{Cardinality}(clients[c1].locks \cap clients[c2].locks) = 0 \\
& \wedge \vee \wedge lock \neq Nil \\
& \quad \wedge clients[lock.client].state = Active \\
& \vee lock = Nil
\end{aligned}$$

Returns a sequence with the head removed

$$Pop(q) \triangleq SubSeq(q, 2, Len(q))$$

Sends a message on the given client's channel

$$\begin{aligned}
Send(m, c) & \triangleq \\
& \wedge messages' = [messages \text{ EXCEPT } ![c] = Append(messages[c], m)] \\
& \wedge messageCount' = messageCount + 1
\end{aligned}$$

Removes a message from the given client's channel

$$\begin{aligned}
Accept(m, c) & \triangleq \\
& \wedge messages' = [messages \text{ EXCEPT } ![c] = Pop(messages[c])] \\
& \wedge messageCount' = messageCount + 1
\end{aligned}$$

Removes the last message and appends a message to the given client's channel

$$\begin{aligned}
Reply(m, c) & \triangleq \\
& \wedge messages' = [messages \text{ EXCEPT } ![c] = Append(Pop(messages[c]), m)] \\
& \wedge messageCount' = messageCount + 1
\end{aligned}$$

Handles a lock request. If the lock is not currently held by another process, the lock is granted to the client. If the lock is held by a process, the request is added to a queue.

$$\begin{aligned}
HandleLockRequest(m, c) & \triangleq \\
& \vee \wedge clients[c].state = Inactive \\
& \quad \wedge Accept(m, c) \\
& \quad \wedge \text{UNCHANGED } \langle clientVars, serverVars \rangle \\
& \vee \wedge clients[c].state = Active \\
& \quad \wedge lock = Nil \\
& \quad \wedge lock' = m @@"client":> c) \\
& \quad \wedge id' = id + 1 \\
& \quad \wedge Reply([type \mapsto LockResponse, acquired \mapsto TRUE, id \mapsto id'], c) \\
& \quad \wedge \text{UNCHANGED } \langle queue, clientVars \rangle \\
& \vee \wedge clients[c].state = Active \\
& \quad \wedge lock \neq Nil \\
& \quad \wedge queue' = Append(queue, m @@"client":> c)) \\
& \quad \wedge Accept(m, c) \\
& \quad \wedge \text{UNCHANGED } \langle lock, id, clientVars \rangle
\end{aligned}$$

Handles a *tryLock* request. If the lock is not currently held by another process, the lock is granted to the client. Otherwise, the request is rejected.

$$\begin{aligned}
\text{HandleTryLockRequest}(m, c) &\triangleq \\
&\vee \wedge \text{clients}[c].\text{state} = \text{Inactive} \\
&\quad \wedge \text{Accept}(m, c) \\
&\quad \wedge \text{UNCHANGED } \langle \text{clientVars}, \text{serverVars} \rangle \\
&\vee \wedge \text{clients}[c].\text{state} = \text{Active} \\
&\quad \wedge \text{lock} = \text{Nil} \\
&\quad \wedge \text{lock}' = m \text{ @@ } (\text{"client"} :> c) \\
&\quad \wedge \text{id}' = \text{id} + 1 \\
&\quad \wedge \text{Reply}([\text{type} \mapsto \text{LockResponse}, \text{acquired} \mapsto \text{TRUE}, \text{id} \mapsto \text{id}'], c) \\
&\quad \wedge \text{UNCHANGED } \langle \text{queue}, \text{clientVars} \rangle \\
&\vee \wedge \text{clients}[c].\text{state} = \text{Active} \\
&\quad \wedge \text{lock} \neq \text{Nil} \\
&\quad \wedge \text{Reply}([\text{type} \mapsto \text{LockResponse}, \text{acquired} \mapsto \text{FALSE}], c) \\
&\quad \wedge \text{UNCHANGED } \langle \text{clientVars}, \text{serverVars} \rangle
\end{aligned}$$

Handles an unlock request. If the lock is currently held by the given client, it will be unlocked. If any client's requests are pending in the queue, the next lock request will be removed from the queue and the lock will be granted to the requesting client.

$$\begin{aligned}
\text{HandleUnlockRequest}(m, c) &\triangleq \\
&\vee \wedge \text{clients}[c].\text{state} = \text{Inactive} \\
&\quad \wedge \text{Accept}(m, c) \\
&\quad \wedge \text{UNCHANGED } \langle \text{clientVars}, \text{serverVars} \rangle \\
&\vee \wedge \text{clients}[c].\text{state} = \text{Active} \\
&\quad \wedge \text{lock} = \text{Nil} \\
&\quad \wedge \text{Accept}(m, c) \\
&\quad \wedge \text{UNCHANGED } \langle \text{clientVars}, \text{serverVars} \rangle \\
&\vee \wedge \text{clients}[c].\text{state} = \text{Active} \\
&\quad \wedge \text{lock} \neq \text{Nil} \\
&\quad \wedge \text{lock}.client = c \\
&\quad \wedge \text{lock}.id = m.id \\
&\quad \wedge \vee \wedge \text{Len}(\text{queue}) > 0 \\
&\quad \quad \wedge \text{LET } \text{next} \triangleq \text{Head}(\text{queue}) \\
&\quad \quad \text{IN} \\
&\quad \quad \wedge \text{lock}' = \text{next} \\
&\quad \quad \wedge \text{id}' = \text{id} + 1 \\
&\quad \quad \wedge \text{queue}' = \text{Pop}(\text{queue}) \\
&\quad \quad \wedge \text{Reply}([\text{type} \mapsto \text{LockResponse}, \text{acquired} \mapsto \text{TRUE}, \text{id} \mapsto \text{id}'], c) \\
&\vee \wedge \text{Len}(\text{queue}) = 0 \\
&\quad \wedge \text{lock}' = \text{Nil} \\
&\quad \wedge \text{Accept}(m, c) \\
&\quad \wedge \text{UNCHANGED } \langle \text{queue}, \text{id} \rangle \\
&\wedge \text{UNCHANGED } \langle \text{clientVars} \rangle
\end{aligned}$$

Expires a client's session. If the client currently holds the lock, the lock will be released and the lock will be granted to another client if possible. Additionally, pending lock requests from the client will be removed from the queue.

$$\begin{aligned}
& \text{ExpireSession}(c) \triangleq \\
& \quad \wedge \text{clients}[c].\text{state} = \text{Active} \\
& \quad \wedge \text{clients}' = [\text{clients} \text{ EXCEPT } ![c].\text{state} = \text{Inactive}] \\
& \quad \wedge \text{LET } \text{isActive}(m) \triangleq \text{clients}'[m.\text{client}].\text{state} = \text{Active} \\
& \quad \text{IN} \\
& \quad \quad \text{IF } \text{lock} \neq \text{Nil} \wedge \text{lock}.\text{client} = c \text{ THEN} \\
& \quad \quad \quad \text{LET } q \triangleq \text{SelectSeq}(\text{queue}, \text{isActive}) \\
& \quad \quad \quad \text{IN} \\
& \quad \quad \quad \quad \vee \wedge \text{Len}(q) > 0 \\
& \quad \quad \quad \quad \quad \wedge \text{lock}' = \text{Head}(q) \\
& \quad \quad \quad \quad \quad \wedge \text{id}' = \text{id} + 1 \\
& \quad \quad \quad \quad \quad \wedge \text{queue}' = \text{Pop}(q) \\
& \quad \quad \quad \quad \quad \wedge \text{Send}([\text{type} \mapsto \text{LockResponse}, \text{acquired} \mapsto \text{TRUE}, \text{id} \mapsto \text{id}'], \text{lock}'.\text{client}) \\
& \quad \quad \quad \quad \vee \wedge \text{Len}(\text{queue}) = 0 \\
& \quad \quad \quad \quad \quad \wedge \text{lock}' = \text{Nil} \\
& \quad \quad \quad \quad \quad \wedge \text{queue}' = \langle \rangle \\
& \quad \quad \quad \quad \wedge \text{UNCHANGED } \langle \text{id}, \text{messageVars} \rangle \\
& \quad \quad \text{ELSE} \\
& \quad \quad \quad \wedge \text{queue}' = \text{SelectSeq}(\text{queue}, \text{isActive}) \\
& \quad \quad \quad \wedge \text{UNCHANGED } \langle \text{lock}, \text{id}, \text{messageVars} \rangle
\end{aligned}$$

Sends a lock request to the cluster with a unique *ID* for the client.

$$\begin{aligned}
& \text{Lock}(c) \triangleq \\
& \quad \wedge \text{clients}[c].\text{state} = \text{Active} \\
& \quad \wedge \text{Send}([\text{type} \mapsto \text{LockRequest}, \text{id} \mapsto \text{clients}[c].\text{next}], c) \\
& \quad \wedge \text{clients}' = [\text{clients} \text{ EXCEPT } ![c].\text{next} = \text{clients}[c].\text{next} + 1] \\
& \quad \wedge \text{UNCHANGED } \langle \text{serverVars} \rangle
\end{aligned}$$

Sends a try lock request to the cluster with a unique *ID* for the client.

$$\begin{aligned}
& \text{TryLock}(c) \triangleq \\
& \quad \wedge \text{clients}[c].\text{state} = \text{Active} \\
& \quad \wedge \text{Send}([\text{type} \mapsto \text{TryLockRequest}, \text{id} \mapsto \text{clients}[c].\text{next}], c) \\
& \quad \wedge \text{clients}' = [\text{clients} \text{ EXCEPT } ![c].\text{next} = \text{clients}[c].\text{next} + 1] \\
& \quad \wedge \text{UNCHANGED } \langle \text{serverVars} \rangle
\end{aligned}$$

Sends an unlock request to the cluster if the client is active and current holds a lock.

$$\begin{aligned}
& \text{Unlock}(c) \triangleq \\
& \quad \wedge \text{clients}[c].\text{state} = \text{Active} \\
& \quad \wedge \text{Cardinality}(\text{clients}[c].\text{locks}) > 0 \\
& \quad \wedge \text{Send}([\text{type} \mapsto \text{UnlockRequest}, \text{id} \mapsto \text{CHOOSE } l \in \text{clients}[c].\text{locks} : \text{TRUE}], c) \\
& \quad \wedge \text{clients}' = [\text{clients} \text{ EXCEPT } ![c].\text{locks} = \text{clients}[c].\text{locks} \setminus \{\text{CHOOSE } l \in \text{clients}[c].\text{locks} : \text{TRUE}\}]
\end{aligned}$$

$\wedge \text{UNCHANGED } \langle \text{serverVars} \rangle$

Handles a lock response from the cluster. If the client's session is expired, the response is ignored. If the lock was acquired successfully, it's added to the client's lock set.

$\text{HandleLockResponse}(m, c) \triangleq$
 $\wedge \vee \wedge \text{clients}[c].\text{state} = \text{Inactive}$
 $\wedge \text{UNCHANGED } \langle \text{clientVars}, \text{serverVars} \rangle$
 $\vee \wedge \text{clients}[c].\text{state} = \text{Active}$
 $\wedge m.\text{acquired}$
 $\wedge \text{clients}' = [\text{clients} \text{ EXCEPT } ![c].\text{locks} = \text{clients}[c].\text{locks} \cup \{m.\text{id}\}]$
 $\wedge \text{UNCHANGED } \langle \text{serverVars} \rangle$
 $\vee \wedge \text{clients}[c].\text{state} = \text{Active}$
 $\wedge \neg m.\text{acquired}$
 $\wedge \text{UNCHANGED } \langle \text{clientVars}, \text{serverVars} \rangle$
 $\wedge \text{Accept}(m, c)$

Receives a message from/to the given client from the head of the client's message queue.

$\text{Receive}(c) \triangleq$
 $\wedge \text{Len}(\text{messages}[c]) > 0$
 $\wedge \text{LET } \text{message} \triangleq \text{Head}(\text{messages}[c])$
 IN
 $\vee \wedge \text{message.type} = \text{LockRequest}$
 $\wedge \text{HandleLockRequest}(\text{message}, c)$
 $\vee \wedge \text{message.type} = \text{LockResponse}$
 $\wedge \text{HandleLockResponse}(\text{message}, c)$
 $\vee \wedge \text{message.type} = \text{TryLockRequest}$
 $\wedge \text{HandleTryLockRequest}(\text{message}, c)$
 $\vee \wedge \text{message.type} = \text{UnlockRequest}$
 $\wedge \text{HandleUnlockRequest}(\text{message}, c)$

Initial state predicate

$\text{Init} \triangleq$
 $\wedge \text{messages} = [c \in \text{Clients} \mapsto \langle \rangle]$
 $\wedge \text{messageCount} = 0$
 $\wedge \text{lock} = \text{Nil}$
 $\wedge \text{queue} = \langle \rangle$
 $\wedge \text{id} = 0$
 $\wedge \text{clients} = [c \in \text{Clients} \mapsto [\text{state} \mapsto \text{Active}, \text{locks} \mapsto \{\}, \text{next} \mapsto 1]]$

Next state predicate

$\text{Next} \triangleq$
 $\vee \exists c \in \text{DOMAIN } \text{clients} : \text{Receive}(c)$

$$\begin{aligned}
&\forall \exists c \in \text{DOMAIN } clients : Lock(c) \\
&\forall \exists c \in \text{DOMAIN } clients : TryLock(c) \\
&\forall \exists c \in \text{DOMAIN } clients : Unlock(c) \\
&\forall \exists c \in \text{DOMAIN } clients : ExpireSession(c)
\end{aligned}$$

The specification includes the initial state predicate and the next state

$$Spec \triangleq Init \wedge \Box[Next]_{\langle serverVars, clientVars, messageVars \rangle}$$

```

\ * Modification History
\ * Last modified Sat Jan 27 02:41:36 PST 2018 by jordanhalterman
\ * Created Fri Jan 26 13:12:01 PST 2018 by jordanhalterman

```